Modeling Parallel Computers as Memory Hierarchies

Bowen Alpern, Larry Carter, and Jeanne Ferrante
IBM T.J. Watson Research Center
Yorktown Heights, N.Y. 10598
(1993 Conference on Programming Models for Massively Parallel Computers)

Abstract

A parameterized generic *model that captures the features of diverse computer architectures would facilitate the development of portable programs.* Specific models *appropriate to particular computers are obtained by specifying parameters of the generic model. A generic model should be simple, and for each machine that it is intended to represent, it should have a reasonably accurate specific model.*

The Parallel Memory Hierarchy (PMH) model of computation uses a single mechanism to model the costs of both interprocessor communication and memory hierarchy traffic. A computer is modeled as a tree of memory modules with processors at the leaves. All data movement takes the form of block transfers between children and their parents. This paper assesses the strengths and weaknesses of the PMH model as a generic model.

1 Introduction

The raw computing power of multiprocessor computers is exploding. The challenge is to create software that can take advantage of this computing power. The diversity of parallel machines, from massively parallel to vector supercomputers, from shared address space to message passing paradigms, from 2D mesh and hypercube to fat tree topologies, makes it difficult to write a single program that runs efficiently on these differing architectures [AC93]. Yet rewriting applications for each new computer is extremely expensive.

One approach to developing portable software is to write a *generic program* — a program that, after machine-specific details have been supplied, results in a good implementation for that machine. A well-known example is the LAPACK linear algebra library [ABetc92], which include a blocking parameter NB that affects the size of submatrices that are passed to the Basic Linear Algebra Subroutines (BLAS).[1] Another class of examples is programs written (perhaps in PVM, Parmacs, or Express) for a collection of sequential processes that communicate by message-passing — in this case, the specific mapping of processes to processors, and perhaps the array partitionings, are machine-dependent.

Perhaps without being aware of it, programmers write generic programs for some *generic model* of computation. The generic model behind LAPACK is a sequential computer with a two-level memory hierarchy. The generic model for many message-passing programs could be the Candidate Type Architecture [S86] which features a finite set of processors in a fixed (but unspecified) bounded degree graph. Undoubtedly the simplest generic model of parallel computers is the Parallel Random Access Machine (PRAM) [FW78]. There is a growing consensus that this model is too simple. Programs which make interprocessor communication explicit generally perform better than programs written for a PRAM [AS91, LS90, NS92].

A generic model defines a class of *specific models*. A specific model appropriate to a particular computer is obtained by specifying parameters[2] of the generic model. Given that a specific model cannot be completely accurate, it is better for it to underestimate rather than overestimate the ability of a computer. It is less wasteful to discard a good algorithm than to implement a poor one, provided that another algorithm can be found which performs well both in the specific model and on the target computer. A specific model will be called *conservative* if it doesn't drastically overestimate any of the machine's capabilities, even if it may underrepresent some. A model

[1] The LAPACK approach also expects the BLAS to be replaced by hand-crafted code for each machine. This suggests that the diversity of real computers cannot be adequately captured by a single blocking parameter.

[2] Here, "parameter" is intended to refer not only to variables representing numbers, but any choice not made explicitly in the model, such as an interconnection topology or whether concurrent reads and/or writes are allowed.

that says each operation takes two weeks is both simple and conservative, but not very useful. Ideally, a specific model should be provably accurate on all possible programs; as this goal appears unattainable, it should be judged by its accuracy on *important* applications, such as those represented by commonly accepted benchmark programs.

A generic model should be simple, and for each machine that it is intended to represent, it should define a specific model that is conservative and reasonably accurate on important benchmarks.

We believe that the key to a good generic model is its ability to adequately capture the costs of data movement in specific architectures. These costs come in two flavors: *interprocessor communication* and *memory hierarchy traffic* (e.g., between registers and cache, or between main memory and disk.)

There are numerous models that address the cost of interprocessor communication [L85, S86, G89, ACS89, CZ89, SLM90, V90, CKetc93, LCHW93], and a vast literature (see [FJetc88, L91]) developing algorithms for different interconnection topologies and parallel architectures. Similarly, there are numerous models for data movement in a memory hierarchy [F72, HK81, AACS87, ACS87, AV88, LTT89, VS90], and many papers show that taking advantage of the memory hierarchy improves performance. These problems are usually attacked separately, even though the efforts required for accomplishing these two objectives are often very similar — they involve breaking a large problem into smaller subproblems that have sufficient locality for the cost of computation to dominate the cost of data movement.

The Parallel Memory Hierarchy (PMH) model [ACF90, ACS90, ACFS93] studied in this paper models both types of data movement costs within the same framework — a tree of memory modules with processors at the leaves. It is most similar to the Fat Tree model [L85], but differs by having memory and message sizes made explicit. It also differs from others models by allowing simultaneous data transfer at different levels of the memory hierarchy.

This paper examines how well the PMH model holds up under the conflicting requirements of simplicity and having reasonably accurate, conservative specific models. This model is formally defined in section 2. Section 3 gives techniques for deriving specific models that accurately reflect common architectural features. Section 4 summarizes the strengths and weaknesses of specific models of several real computers. The final section assesses the PMH model as a generic model of parallel computation.

2 The PMH Model

A parallel computer is modeled as a tree of modules. Each child is connected to its parent by a unique channel. Modules hold data; leaf modules also perform computation — arithmetic operations, comparisons, indexing, branching, and so on. Data in a module is partitioned into *blocks*. A block is the unit of transfer on the channel connecting a module to its parent. All of the channels can be active at the same time, although two channels cannot simultaneously move the same block.[3]

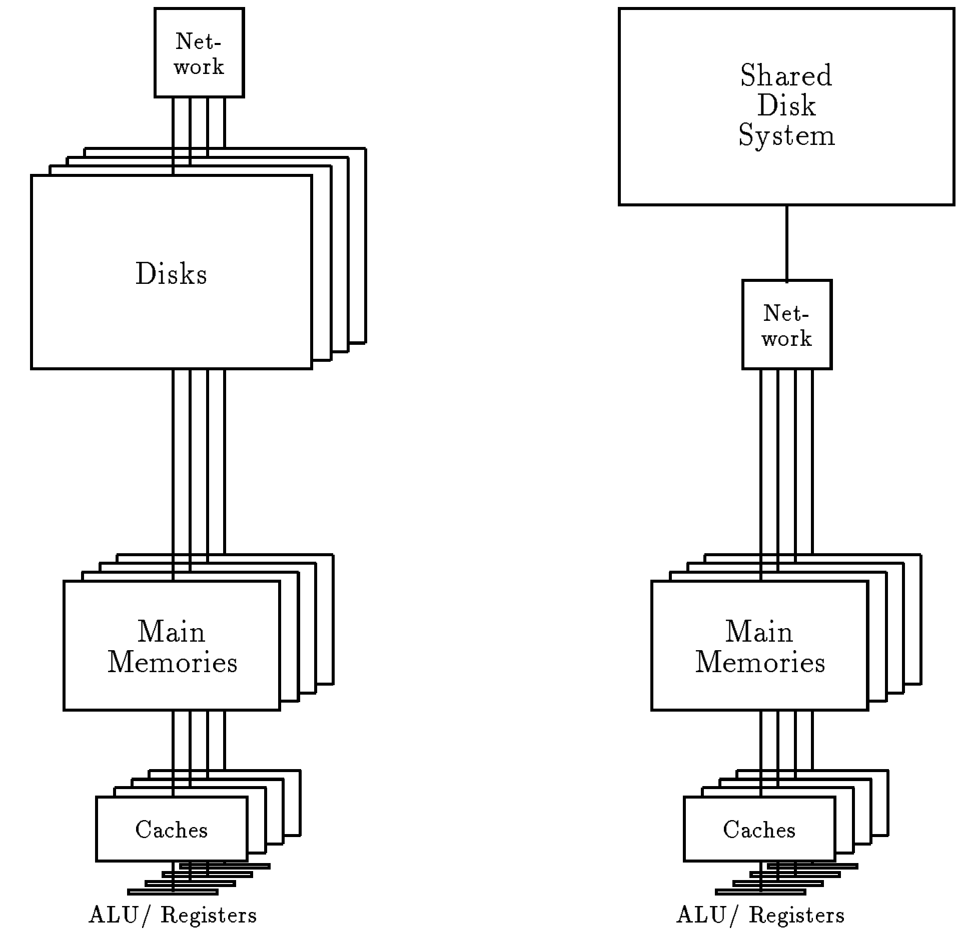
The model has four parameters for each module m: the *blocksize* s_m tells how many bytes there are per block of m; the *blockcount* n_m tells how many blocks fit in m; the *childcount* c_m tells how many children m has; and the *transfer time* t_m tells how many cycles it takes to transfer a block between m and its parent. Generally, all modules at a given level of the tree will have the same parameters.

The p-processor PRAM model [FW78] is the special case of the PMH model with only two levels — a root (representing all of memory) with p children, each having blocksize 1, blockcount some small constant, and transfer time 1. The use of a tree to model a parallel computer's communication structure is a compromise between the simplicity of the PRAM model, and the accuracy of an arbitrary graph structure. Leiserson [L85] gives an interesting justification for the validity of a tree model based on physical considerations. The H-PRAM model [HR92] is another model that assumes a tree organization for the processors. Our model differs from these other models by allowing data to be stored in the intermediate nodes of the tree. This allows us to use the same structure to model the memory hierarchy, which in our experience [AABCH91, TAC92, ACFS93] is crucially important for developing high-performance programs.

3 Choosing a Specific Model

A three-step approach can be used to derive a specific PMH model for a given architecture. First, choose an overall tree structure to reflect the data transfer bandwidths among the processors and between proces-

[3] Thus, the PMH is an Exclusive-Read, Exclusive-Write (EREW) model. This reflects our observation that on most computers, broadcasts are more expensive than a CREW model would suggest.



(a) Low-Bandwidth Network

(b) High-Bandwidth Network

Figure 1: Specific models for a network of workstations

sors and memory. Next, choose blocksizes to model communication delays. Finally, choose blockcounts to model memory sizes and any communication degradation caused by contention. These steps are elaborated in the following subsections.

3.1 Modeling Overall Structure

To model a particular computer, one must chose a tree structure that models the machine's communication capabilities and memory hierarchy. The most important consideration is that this structure reflect the data bandwidths between different architectural components (processors, memory, disks, communication controllers, etc.) of the computer. The fact that the PMH is a tree implies that each channel is the only connection between two subgraphs. If this channel has a low bandwidth,[4] then it is impossible to model a high-bandwidth data path between two components that are not in the same subgraph.

This consideration suggests a basic strategy for choosing the overall tree structure and bandwidths. Create a graph G to represent the computer using nodes for significant architectural components and edges to represent data paths between components. The weight of each edge should be the bandwidth of the represented data path. Create a leaf module in the PMH model M for each processor node in G. Now set

[4]The *bandwidth* of the channel from module m to its parent, measured in bytes per cycle, is s_m/t_m.

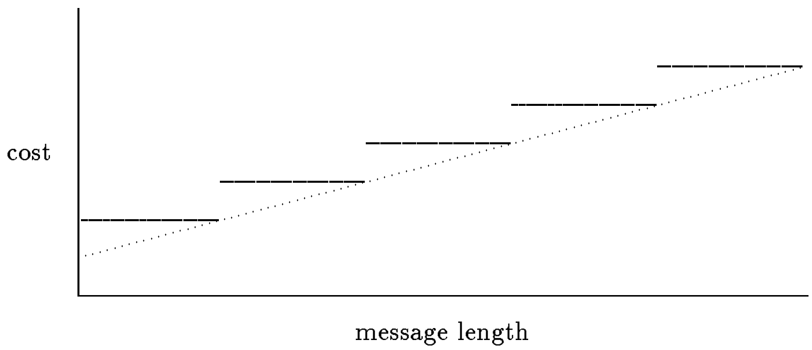


Figure 2: Modeling Latency and Bandwidth in the PMH model.

the "threshold bandwidth" b to be the maximum weight of any edge to a processor node. In G, merge all pairs of nodes connected by edges of weight b or higher;[5] additionally, create a parent module in M to represent the subgraph of G that was merged with each processor module. Thus, whenever processor nodes are merged in G, the parent modules introduced in M will have multiple children. Repeat this procedure — choosing b, merging nodes of G, and creating a new level of M — until G is a single node.

As an example, consider a network of workstations, each with a local cache, memory and disk, that are connected by a Token Ring. The workstations' caches and memories, having higher bandwidths to the processors than the Token Ring, will be represented as a chain above the leaf nodes as in Figure 1.

There are several ways to model the connection between the workstations. If the bandwidth from one processor to its disk is greater than its bandwidth to the network, then we model the network as a root module attached to the separate disk modules as in Figure 1(a). A message sent from workstation X to workstation Y is modeled as a stream of blocks transferred up to the module representing X's disk, then to the root module and then down to Y's disk and main memory.

Alternatively, if the bandwidth between two processors is greater than the bandwidth to disk, then we get the structure shown in Figure 1(b). The root of the tree represents the combined storage capacity of all the disks, and it has a single child representing the network. This model hides the differences between a workstation's local disk and the other workstations' disks. However, this inaccuracy is not too significant, since the high-bandwidth network allows a processor to access all the disks at comparable costs.

3.2 Modeling Latency

It is common to model the communication capacity between components of a distributed system in terms of a *latency* L and a *bandwidth* B. Under this "L–B" model, sending or receiving a message of length N takes $L + N/B$ time. The PMH model captures the same phenomenon with its two parameters, blocksize s and transfer time t. This subsection explores how closely the PMH can approximate the L–B model.[6]

In the PMH model, the cost of communicating a message is always a step function of the message length. Parameters must be chosen so that the step function they determine is a close approximation to the linear L–B cost function. First, consider a 2-level PMH model. A natural parameter choice is $s = L * B$ and $t = L$. This blocksize s corresponds to Hockney's $n_{\frac{1}{2}}$, the length for which the overhead due to latency exactly balances the time spent in useful communication [H91]. Figure 2 compares the two models; the dotted line is the L–B cost function, and the step function is the cost under the PMH model.[7] The relative error $(\frac{cost_{PMH} - cost_{LB}}{cost_{LB}})$ is non-negative, it never exceeds 1, and it is asymptotically zero. Thus this PMH model gives a conservative and reasonable approxima-

[5] Assign weights to edges after merging nodes by considering the aggregate bandwidth of the data paths. Depending on the computer being modeled, this may be *less* than the sum of the weights of the merged edges.

[6] The L–B model itself is only an approximation to the true communication costs. More refined models make the latency dependent on the distance between processors, or reflect the partitioning of messages into blocks that are sent in a pipelined manner. Since these refinements are in the direction of the PMH model, we would expect the PMH models to have *improved* accuracy.

[7] A message of length at most s requires $2t$ cycles, since a block must be moved up and then down; but multiblock messages take only an additional t cycles per block due to pipelining.

	level	childcount	blockcount	blocksize	transfer time
l		c_l	n_l	s_l	t_l
3	Data Network	64	512	74 bytes	
2	Group of 16	4	16	74 bytes	37
1	Group of 4	4	8	74 bytes	74
0	Processing Node	0	453438	74 bytes	148

Figure 3: PMH parameters for the CM-5.

tion.[8]

Though not shown here, a three-level hierarchy model allows agreement with the L–B curve both for short messages and asymptotically, and can further improvement the relative error.

Returning to the example of the network of workstations, we are confronted with an unfortunate but unavoidable aspect of the PMH model: when the structure of the PMH tree is chosen to accurately model the bandwidths (as in the earlier subsection), then the model may be inaccurate for communication latencies or disk access times. The PMH model cannot simultaneously represent a high bandwidth disk with a long seek time and a low bandwidth network connection with very low communication delay. This is a good example of how the desire to keep a generic model simple conflicts with the accuracy of its explicit models.[9]

The above discussion only considers communication costs between a pair of processors. The next section explores the more general case, where many processors are sending messages simultaneously.

3.3 Modeling Contention

For some architectures, there will be one communication bandwidth B_1 between pairs of processors on a lightly loaded bus or network, but a lower bandwidth B_2 when many processors are communicating at the same time. Suppose one wished to model such a machine (or some portion of it) by a two-level PMH. One possibility is simply to give each child module a bandwidth of B_2 with the parent module. This conforms with the desire to model machines conservatively.[10]

An equally conservative but more accurate (though less simple) possibility is to choose the blockcount of the parent module to be cB_2/B_1 where c is the number of children, and choose s and t for the children so that $s/t = B_1$. Thus, a single pair of children can communicate with bandwidth B_1. However, the narrow parent module limits the system to cB_2/B_1 simultaneous communications each at a bandwidth of B_1, so the average communication bandwidth of a heavily loaded network is B_2.

4 Modeling Real Computers

There are methodological difficulties in even defining what it means to compare a specific model to a "real computer". To circumvent this problem, we define a *performance model* to be another model, one motivated by a real machine, that is an abstraction of the machine's performance characteristics.

4.1 The CM-5

The performance model of interprocessor communication we consider is as follows: The CM-5 (without vector processors) is 4-ary Fat Tree with 1024 processing nodes. In the absence of other communication, two processing nodes communicate with bandwidth $\frac{1}{2}$ byte/cycle and latency $16h+132$ cycles, where h is the

[8] A slightly less conservative model has lower relative error but the same asymptotic behavior. Taking $t = \frac{\sqrt{5}-1}{2}L$ and $s = tB$ improves the maximum relative error to $\sqrt{5} - 2 \approx .24$.

[9] It does not necessarily mean the PMH is a poor generic model. To fault a generic model, one would have to find a commercially significant architecture which can only be modeled poorly, and demonstrate that this inaccuracy leads to the design of poor algorithms and programs. This is beyond the scope of this paper.

[10] The fact that it overestimates the communication time when there is little communication may not affect the overall predicted time for computation-bound applications.

height of their lowest common ancestor.[11] When all processors are communicating in pairs simultaneously, the bandwidth between two nodes is $\frac{1}{2}$ if $h = 1$, $\frac{1}{4}$ if $h = 2$, and $\frac{1}{8}$ otherwise.

Using the conservative two-level method described in Section 3.2 and choosing a 4-level PMH tree to reflect the high-contention bandwidths accurately leads to the specific model of Figure 3. The predicted cost of a single point-to-point communication is always one to three times the true cost, the maximum relative error of 184% occurring for a one-byte message with $h = 3$.

A PMH model can capture many architectural features more completely than other models. However, it is imperfect. One feature of the CM-5 that is not modeled particularly well by a PMH is the control network. The machine's hardware can, for instance, perform a parallel-prefix operation on a set of numbers — one on each processor — in perhaps 200 cycles. Our model's prediction is off by about a factor of 10.

4.2 KSR1

Briefly, the KSR1 architecture [KSR91] has a two-level hierarchy of rings, a single address space implemented on physically distributed memory, and high-performance RISC processors with large register sets and local caches. The PMH model is quite accurate for the bandwidth and latency characteristics of the interprocessor communication, as well as the memory hierarchy dimensions, with a couple of caveats:

- The bandwidth *into* the local memory of a processor is limited by its ability to have only 4 separate requests for 128-byte "subpages" outstanding simultaneously. However, the bandwidth *out* to other processors is much higher. A conservative PMH model must make both equal to the smaller bandwidth. Thus, the model underrepresents the ability of the machine to distribute data from one processor to all others.

- When the KSR1 brings a subpage into a local memory, it reserves space for the full page that the subpage belongs to. Further, there are restrictions on which addresses can simultaneously reside in local memories and caches due to the limited set-associativity of these memories. If these constraints are modeled conservatively, the machine's local memory and cache capacities are severly underrepresented. An alternative, following common practice, is to ignoring such problems, hoping that applications will have enough locality of reference to make use of the full memory sizes.

[11] This formula is taken from a study of the communication speeds using the Active Message communication protocol [CKetc93, VCGS92].

4.3 Other Computers

The hardest common architecture for the PMH to model is a 2-dimensional mesh. The PMH must model the mesh as an k-ary tree (for k some small power of 2) in order to reflect the $O(\sqrt{P})$ bisection bandwidth out of a square of P processors. Unfortunately, there will be processors that are adjacent in the mesh, but far separated in the PMH tree. The communication latencies between adjacent processors must necessarily be inaccurate, and a conservative PMH model may not be able to model systolic array algorithms well. However depending on details of a particular mesh, a model that assigns a large blocksize to the processor modules but a small blocksize and transfer cost to the interior modules may be reasonably accurate.

The easiest architecture to model might well be ones, like IBM's SP-1, that consist of a collection of RISC processors with local memory, connected by a switch that allows arbitrary point-to-point communication patterns. For such machines, a model like that of Figure 1(b) is appropriate.

5 Conclusion

The PMH model with its zoo of parameters (four per module) at first appears anything but simple. However, its tree structure is simpler than a model which allows more complex graph structures, and it reflects the Fat Tree organization that appears to be gaining in popularity. Generic divide-and-conquer programs can be written for the PMH model by writing a routine for one parameterized memory module that is invoked recursively. Another "simplicity" of the PMH model is that the same techniques can be used for improving performance of disk, cache, register, and network usage.

On the other hand, many simpler generic models rely on the computer's virtual memory system or optimizing compilers to support the illusion of large memories. Unfortunately, when good performance on large problem instances is desired, one cannot pretend that the main memory of each processor is infinitely large; the programmer must sometimes exercise explicit control over the data movement. Further, on complicated programs, optimizing compilers do not always handle communication, cache, or registers well. It is for these reasons that we advocate a model that makes the memory hierarchy explicit.

Experience developing high-performance programs suggests that the PMH is successful at modeling *memory hierarchy traffic.* This paper has focused on *interprocessor communication.* Section 3.2 shows that by choosing the blocksize and transfer cost parameters judiciously, the speed of sending a single message from one processor to another can be modeled fairly accurately, even though the model entails fixed-sized transfers that must go up and down a tree. Section 3.3 shows that contention can be handled by limiting the blockcount of the module that represents a bus.

Models of several of the newest generation of supercomputers are summarized here. Many, but not all, important features of the CM-5 and the KSR1 can be easily represented using the PMH model. For instance, a conservative PMH model of the idealized CM-5 data network has a worst-case relative error of 184%, but it models the control network much less accurately.

We believe it is worthwhile to develop a method for comparing generic models that is more accurate than given by big-O results. The exercise (in the footnote of section 3.2) of reducing the maximum relative error in a specific model for latency and bandwidth from 1.0 to 0.24 is probably overkill. But with a good generic model, it should be possible to quantify the inaccuracy of specific models for aspects of real machines. This would help formalize comparison among proposed generic models. We hope this paper will encourage others to join in the development of better models with the goal of reaching consensus on a generic model of parallel computation.

Acknowledgements

We thank Shahzad Bhatti, Klaus Schauser, and Liddy Shriver for asking provoking questions and suggesting improvements.

References

[AACS87] Aggarwal, A., B. Alpern, A. K. Chandra, and M. Snir, "A Model for Hierarchical Memory," *Proc. 19th Symp. on Theory of Comp.*, May 1987, pp. 305-314.

[ACS87] Aggarwal, A., A. K. Chandra, and M. Snir, "Hierarchical Memory with Block Transfer," *Proc 28th Symp. on Foundations of Comp. Sci.*, October 1987, pp. 204-216.

[ACS89] Aggarwal, A., A. K. Chandra, and M. Snir, "On Communication Latency in PRAM Computations," *Proc 1989 ACM Symp. on Parallel Algorithms and Architectures*, June 1989, pp. 11-21.

[AV88] Aggarwal, A. and J. Vitter, "IO Complexity of Sorting and Related Problems," *CACM*, September 1988, pp. 305-314.

[AHU74] Aho, A., J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Massachusetts, 1974.

[AABCH91] Almasi, G., B. Alpern, L. Berman, L. Carter, and D. Hale, "A Case-study in Performance Programming: Seismic Migration ," *High Performance Computing II*, North Holland, October 1991.

[AC93] Alpern, B. and L. Carter, "Towards a Model for Portable Parallel Performance: Exposing the Memory Hierarchy," *Workshop on Portability and Performance for Parallel Processing*, Southampton, England, July 1993, to be published by Wiley.

[ACS90] Alpern, B., L. Carter, and T. Selker, "Visualizing Computer Memory Architectures," *IEEE Visualization '90 Conference*, October 1990.

[ACF90] Alpern, B., L. Carter, and E. Feig, "Uniform Memory Hierarchies," *Proc 31st Symp. on Foundations of Comp. Sci.*, October 1990.

[ACFS93] Alpern, B., L. Carter, E. Feig, and T. Selker "The Uniform Memory Hierarchy Model of Computation," *Algorithmica*, to appear.

[ABetc92] Anderson, E., Z. Bai, C. Bishof, J. Demmel, J.Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, amd D. Sorensen, *LAPACK Users' Guide*, SIAM, Philadelphia, 1992.

[AS91] Anderson, R. and L. Snyder, "A Comparison of Shared and Nonshared Memory Models of Parallel Computation," *Proceedings of the IEEE*, April 1991, pp. 480-487.

[CZ89] Cole, R. and O. Zajicek, "The APRAM: Incorporating Asynchrony in the PRAM Model," *Proc 1989 ACM Symp. on Parallel Algorithms and Architectures*, June 1989, pp. 169-178.

[CKetc93] Culler, D., R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. von Eicken, "LogP: Towards a Realistic Model of Parallel Computation," *4th Symp. on Principles and Practice of Parallel Programming*, May 1993.

[FW78] Fortune, S. and J. Wyllie, "Parallelism in Random Access Machines," *Proc. ACM STOC*, 1978, pp. 114-118.

[F72] Floyd, R. W., "Permuting Information in Idealized Two-Level Storage," *Complexity of Computer Computations*, Plenum Press, New York, 1972, pp. 105-109.

[FJetc88] Fox, G. M. Johnson, G. Lyzenga, S. Otto, J. Salmon, D. Walker, *Solving Problems on Concurrent Processors*, Volume I, Prentice Hall, Englewood Cliffs, 1988.

[G89] Gibbons, P., "A More Practical PRAM Model," *Proc. SPAA,* 1989, pp. 158-168.

[HR92] Heywood, T. and S. Ranka, "A practical Hierarchical Model of Parallel Computation I. The Model," *Journal of Parallel and Distributed Computing*, Vol. 16, 1992, pp. 212-231.

[H91] Hockney, R.W., "A Framework for Benchmark Performance Analysis," *Proc. 2nd Euroben Workshop,* Sept 1991.

[HK81] Hong, J-W. and H. T. Kung, "I/O Complexity: The Red-Blue Pebble Game," *Proc. 13th. Symp. on Theory of Comp.*, May 1981.

[KSR91] Kendall
Square Research Corporation, "KSR1 Principles of Operation," October, 1991.

[LTT89] Lam, T., P. Tiwari, and M. Tompa, "Tradeoffs Between Communication and Space," *Proc. 21th Symp. on Theory of Comp.*, May 1989, pp. 217-226.

[LCHW93] Larus, J., S. Chandra, M. Hill and D. Wood, "CICO: A Shared-Memory Programming Performance Model," *University of Wisconsin at Madison*, 1993, to appear.

[L85] Leiserson, C. E., "Fat-trees: Universal Networks for Hardware-efficient Supercomputing," *IEEE Trans. on Computing*, 1985, pp 892-901.

[L91] Leighton, F., *Introduction to Parallel Algorithms and Architectures: Networks and Algorithms*, Morgan-Kaufmann, 1991.

[LS90] Lin, C. and L. Snyder, "A Comparison of Programming Models for Shared Memory Multiprocessors," *Proc. ICPP Vol II*, August 1990, pp. pp. 163-170.

[NS92] Ngo, T. and L. Snyder, "On the Influence of Programming Models on Shared Memory Computer Performance," *Proc. SHPCC*, 1992, pp. pp. 284-291.

[S86] Snyder, L., "Type Architectures, Shared Memories, and the Corollary of Modest Potential," *Annu. Rev. Comput. Sci. 1*, 1986, pp. 289-317.

[SLM90] Scott, M., T. LeBlanc, and B. Marsh, "Multi-model Parallel Programming in Psyche," *ACM Symp. on Principles and Practice of Parallel Programming*, 1990, pp. 70-78.

[TAC92] Thomborson, C., B. Alpern and L. Carter, "Rectilinear Steiner Tree Minimization on a Workstation," *DIMACS Workshop on Computational Support For Discrete Mathematics*, March 1992, Also IBM RC 17680.

[V90] Valiant, L. G., "A Bridging Model for Parallel Computation," *CACM*, August 1990, pp. 103-111.

[VS90] Vitter, J.S. and E.A.M. Shriver, "Optimal Disk I/O with Parallel Block Transfer," *Proc. 22th Symp. on Theory of Comp.*, May 1990.

[VCGS92] Von Eicken, T., D. Culler, S. Goldstein, and K. Schauser, "Active Messages: A Mechanism for Integrated Communication and Computation," *Proc. 19th Int'l. Symp. on Computer Architecture*, May 1992.